Tute Answer - Week 4

Exercise 01

```
interface IDisplay {
        void print(); // Print in one line
        void printDetails(); // Print in multiple Lines
}
interface IInput {
        void input();
}
```

Two interfaces are given in the question. [Interface is a pure abstract class where you cannot write any implementations for the methods (all the methods are by default abstract methods) , only the method declarations will be there in the interface and the class which is implementing those interface should override those methods]

Exercise 01

a) Book class should be created with 3 attributes.

```
public class Book
{
    int bookId;
    String title;
    String publisher;
}
```

which uses the interfaces IDisplay and IInput . Now we have to make this book class to use these two interfaces, to do that we have to use **implements** keyword.

We can separate the interfaces using comma **,**

First we will implement the IDisplay interface as bellow,

```
public class Book implements IDisplay

        {
             int bookId;
             String title;
             String publisher;
        }
```

Now Book class is implementing IDisplay interface, now we must override the methods in the IDisplay interface (print() and printDetails())

Overriding – same method signature having different implementation in a different class.

```
public class Book implements IDisplay

        {
             int bookId;
             String title;
             String publisher;
        }

@Override
    public void print() {

        System.out.println("Book ID   " + this.bookId +
                " Book title   " + this.title +
                " Publisher    " + this.publisher);

    }
    @Override
    public void printDetails() {
```

```
        System.out.println("Book ID   " + this.bookId );
        System.out.println("Book title " + this.title );
        System.out.println("Publisher " + this.publisher);

    }
```

Now will implements the other interface IInput , as follows , use comma to separate interface names.

```
    public class Book implements IDisplay, IInput{ }
```

Now your Book class is implementing IInput interface, in IInput interface we have a method called Input(). So now in Book class we have to override that input method.

So finally

```
import java.util.Scanner;

public class Book implements IDisplay,IInput{

    int bookId;
    String title;
    String publisher;
    @Override
    public void input() {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the book ID");
         this.bookId = sc.nextInt();

        System.out.println("Enter the book title");
        this.title = sc.next();

        System.out.println("Enter the publisher");
        this.publisher = sc.next();

    }
```

```
    @Override
    public void print() {

        System.out.println("Book ID   " + this.bookId +
                  " Book title   " + this.title +
                  " Publisher    " + this.publisher);

    }

    @Override
    public void printDetails() {

        System.out.println("Book ID   " + this.bookId );
        System.out.println("Book title " + this.title );
        System.out.println("Publisher " + this.publisher);

    }

}
```

Now Book class should override totally 3 methods(2 methods from IDisplay and 1 method from IInput). If atleast 1 method is not implemented in the Book class then the Book class should be defined as Abstract class.

b) **Similarly implement the student class as well.**

```
public class Student implements IDisplay,IInput{

    String studentId;
    String name;

public void input() {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the Student ID");
        this.studentId = sc.next();

        System.out.println("Enter the name");
        this.name = sc.next();

    }
    @Override
```

```
    public void print() {
          System.out.println("Student ID " + studentId + " Name" + name );

    }
    @Override
    public void printDetails() {
          System.out.println("Student ID " + studentId );
          System.out.println("Name " + name );
    }
}
```

c) **Create objects of the Book and Student in the main method**.

Main method should be created in the separate class; let's create a class called Demo for that.

```
public class Demo {

      public static void main(String[] args) {

      // Create objects of the Book
      Book bo = new Book();

      // Create objects of the Student
      Student so =  new Student();

      //create variable from IDisplay
      IDisplay ref1 ;

      //create variable from IInput
      IInput ref2;

      // ref1 variable is now pointing to Book object . ref1 object type is
Book, reference type is IDisplay
      ref1 = new Book();
      ref1.printDetails();  now using ref1 we can call the printDetails()

      // ref2 variable is now pointing to Student object . ref2 object type
is Student,  type is IInput
      ref2 = new Student();
      ref2.input();   now using ref2 we can call the input() method


      // we can call all the 3 methods using bo Book object.
      bo.input();
      bo.print();
      bo.printDetails();
```

```
            // we can call all the 3 methods using so Student object.
            so.input();
            so.print();
            so.printDetails();

        }

       }
```

Exercise 2

Create a class called Account with 3 attributes AccountNumber, name and balance

```
public class Account {

       String accountNo, name;
       double balance;

}
```

Implement a Deposit() method to deposit money. The amount deposited should update the balance

```
public class Account {

       String accountNo, name;
       double balance;

       void deposit(double amount)    //amount is passed as an argument through the method
       {
              balance = balance + amount;
        }
}
```

Implement a constructor to get values to the Account class.

```
public class Account {

       String accountNo, name;
       double balance;
       //overloading Constructor
       public Account(String accountNo, String name, double balance) {

              this.accountNo = accountNo;
              this.name = name;
              this.balance = balance;
       }
       void deposit(double amount)
       {
```

```
            balance = balance + amount;
        }
}
```

Have an abstract method called calculateInterest() which returns a double value

Until now Account class is a normal concrete class, now we are going to declare an abstract method inside this class. When you have at least 1 abstract method in a class, then the whole class will become abstract. So now we need t

```
public abstract class Account {

        String accountNo, name;
        double balance;

        abstract double  calculateInterest();


        //overloading Constructor
        public Account(String accountNo, String name, double balance) {

                this.accountNo = accountNo;
                this.name = name;
                this.balance = balance;
        }
        void deposit(double amount)
        {
                balance = balance + amount;
         }
}
```

Implement a method to display() the account details

```
public abstract class Account {

        String accountNo,name;
        double balance;


        abstract double  calculateInterest();

        public Account(String accountNo, String name, double balance) {
```

```
            this.accountNo = accountNo;
            this.name = name;
            this.balance = balance;
      }

      void deposit(double amount)
      {
            balance = balance + amount;
      }
       //printing all the three detail
       void display()
       {
             System.out.println("Account No" + accountNo);
             System.out.println("Name " + name);
             System.out.println("Balance " + balance);
       }

}
```

FixedDepositAccount Class - Implement a new class called FixedDepositAccount which extends the Account class.

```
public class FixedDepositAccount extends Account { }
```

It should have a new property called interestRate and Interest

```
public class FixedDepositAccount extends Account {

      double interestRate , Interest ;
 }
```

Write a setter and getter for the interestRate

Set method is used to assign values to an attribute. So you **need to pass the value you are going to assign through the** parameter and since we are going to use this method to perform only the assignment operation this method doesn't need to return anything so always set method can have the return type as **void**.

```
public void setInterest(double Finterest) {
            Interest = Finterest;
}
```

Or

```
public void setInterest(double Interest) {
            this. Interest = Interest;
}
```

Get method is used to return the value of an attribute , so we don't need to pass any arguments for the method, but we need to consider the return type. Return type of the method should be the data type of the attribute. Example in here Interest is double variable so get method should return double.

```
public double getInterest() {
            return Interest;
```

```
}
```

Now your program will look like this,

```
public class FixedDepositAccount extends Account {

    double interestRate , Interest ;

    public double getInterest() {
        return Interest;
    }

    public void setInterest(double interest) {
        Interest = interest;
    }
}
```

You can see there is an error in the beginning of the class . What is that and why?

Because Account class is an abstract class, which has 1 abstract method. The **classes which are extending the abstract class should override all the abstract methods in the parent. If not child class should be defined as an abstract.**

So here Account class in an abstract class having **calculateInterest() – abstract method.** So FixedDepositAccount class should override the **calculateInterest() method, if not FixedDepositAccount class also should be defined as Abstract class.**

So now we will implement the **calculateInterest() method according to part (i)**

```
double calculateInterest() {
            // TODO Auto-generated method stub

            Interest = balance * interestRate/100 ;
            return       Interest;
}
```

Finally

```java
public class FixedDepositAccount extends Account {
    double interestRate , Interest ;

    public double getInterest() {
        return Interest;
    }

    public void setInterest(double interest) {
        Interest = interest;
    }

    @Override
    double calculateInterest() {
        // TODO Auto-generated method stub

        Interest = balance * interestRate/100 ;
        return  Interest;
    }

}
```

SavingsAccount Class

Implement a new class called SavingAccount which inherits the FixedDepositAccount class

```java
public class SavingAccount       extends FixedDepositAccount{



}
```

Implement a withdraw() method that allows you to withdraw money from the SavingsAccount

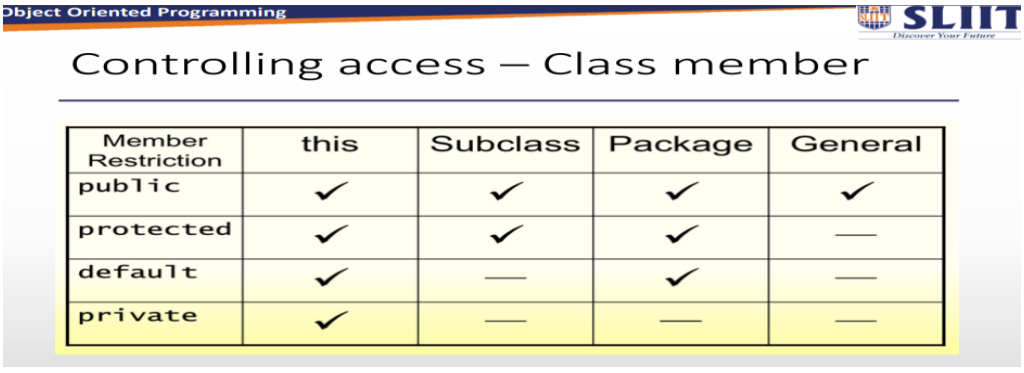
```java
public class SavingAccount       extends FixedDepositAccount{

      void withdraw(double amount)
      {
            balance = balance - amount;
      }

}
```

Create objects from the FixedDeposit and SavingAccount. Call the Deposit() and withdraw() methods (Only SavingsAccount have withdrawals)

To create objects and call the methods, We will create a different class with main method

```
public class MyMain {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Account acc1 = new FixedDepositAccount();  //fixed deposit object
		SavingAccount acc2 = new SavingAccount();   //savingAccount object

		acc1.deposit(10000);

		acc2.deposit(5000);
		acc2.withdraw(20.0);


		acc2.display();

	}

}
```

Note – if you are going to consider access modifiers, now all the attribute in the above classes are default. But you can have the answer like this as well. Because accountNo, name we are using only inside Account class so private in Account class, but balance is declared in Account class but to be used in subclasses as well , so it should be protected. InterestRate and Interest should be declared in FixedDepositAccount class and also needed in subclass so they should be protected.



```java
public abstract class Account {

    private  String accountNo,name;
    protected double balance;


    abstract double  calculateInterest();

    public Account()
    {
    }

    public Account(String accountNo, String name, double balance) {

        this.accountNo = accountNo;
        this.name = name;
        this.balance = balance;
    }

    void deposit(double amount)
    {
        balance = balance + amount;
    }

     void display()
     {
         System.out.println("Account No" + accountNo);
         System.out.println("Name " + name);
         System.out.println("Balance " + balance);
     }

}
```

```java
public class FixedDepositAccount extends Account {
    protected double interestRate , Interest ;

    public double getInterest() {
        return Interest;
    }

    public void setInterest(double interest) {
        Interest = interest;
    }

    @Override
    double calculateInterest() {
        // TODO Auto-generated method stub

        Interest = balance * interestRate/100 ;
        return  Interest;
    }

}
```

```java
public class SavingAccount  extends FixedDepositAccount{

    void withdraw(double amount)
    {
        balance = balance - amount;
    }

    double calculateInterest()
    {
        Interest = balance * interestRate/100/12;
        return Interest;
    }

}
```